UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
INSTITUTE OF COMPUTER SCIENCE

Study and visualiser of different clustering methods for density-based clustering

Project report

Martin Loginov (martin.loginov@ut.ee)
Hans Mäesalu (hansmaesalu@gmail.com)
Sven Aller (sven.aller@ut.ee)

TARTU 2011

Introduction

Data clustering is an important data exploration techique with many applications in data mining. There are many different types of methods for clustering data: centroid based clustering, hierarchical clustering, density based clustering etc. In our project we will focus only on density based clustering. In density-based clustering, clusters are defined as areas of higher density than the remainder of the data set. Objects in these sparse areas - that are required to separate clusters - are usually considered to be noise and border points.

The main goal of our project is to develop a simple clustering application that would allow users to run three different density-based clustering algorithms on different datasets consisting of 2D point data. We have chosen to implement the following three algorithms for our application: DBSCAN, DCBOR and SNN.

The first chapter will give a brief overview of each algorithm, describing the main concepts of how they work. The next chapter describes the application itself and some of the implementation details behind it. It also has a full description of the application functionality and can be used as a reference. In the last chapter we describe how our implementations of the three algorithms perform on some example datasets and try to make some comparison between them.

Algorithms

DBSCAN

DBSCAN, density-based spatial clustering of applications with noise, is density based clustering algorithm developed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xuaiweu Xu [1]. Unlike partitioning or hierarchical clustering algorithms, DBSCAN is based on density based notion of clusters, this means that clusters are formed in most dense regions. It is designed to be easy to use, requires minimal domain knowledge for clustering, and can discover clusters of arbitrary shape.

DBSCAN needs 2 parameters as input: *eps*, neighborhood radius, and *minPts*, the minimum number of points required to form a cluster. DBSCAN starts with a random point and retrieves it's density-reachable neighborhood. If it contains at least *minPts* neighbors then a new cluster is started, otherwise point is marked as noise. Noise points can be later added to other clusters. If point is part of a cluster then all points in its neighborhood are also marked as part of this cluster. This continues until the whole cluster is found, then a new unvisited point is taken and processed like previous points.

DBSCAN does not require user to know the number of clusters beforehand. It can also find clusters of different shapes and has notion of noise.

With appropriate indexing structure that can find point neighborhood in *O(log n)* time, for example R*-tree, the overall run-time complexity of DBSCAN would be *O(n*log n)*, since neighborhood has to be found for each point once. Our implementation does not use accelerating index structure, so our implementation time complexity is $O(n^2)$.

DCBOR

This chapter is based on a research paper by A. M. Fahim, G. Saake, A. M. Salem, F. A. Torkey and M. A. Ramadan [6]

DCBOR stands for Density Clustering Based on Outlier Removal. It is an enhanced version of the well known single link clustering algorithm [8]. This algorithm provides outlier detection and data clustering simultaneously. The algorithm consists of two phases. During the first phase, it finds the k-nearest neighbors of every datapoint and removes the outliers from the data set. During the second phase, it uses the single link algorithm with simple modification to discover the genuine clusters.

The main idea of the outlier removal phase of the algorithm is to remove the lowest density points from the data. The density of a point is computed according to the following two functions: the influence function represents the impact of point *x* on point *y* as the Euclidean distance between them:

$$INF(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

The density function for a point x is defined as the summation of (influence functions within the k nearest neighbors) distances between the point x and the k nearest neighbors:

$$DEN(x, y_1, y_2, ..., y_k) = \sum_{i=1}^{k} INF(x, y_i)$$

The authors claim that the definition of density, based on the summation of distances of the k nearest neighbors, is better than counting the points within the Eps-neighborhood radius like DBSCAN does. Consider the following example as in Figure 1.

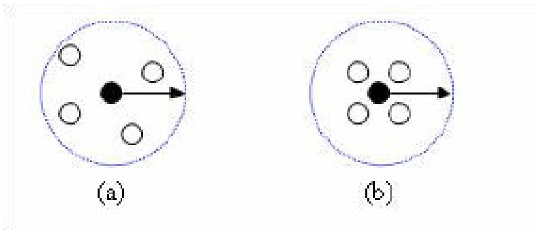


Figure 1. An example of two points Eps-neighborhoods.

As we see in Figure 1.a and 1.b both black points have four points in their neighborhoods (the Eps is the same, represented by black arrow) but in Figure 1.b the black point has a higher density. The summation of distances to nearest neighbors reflects this fact accurately. But based on the Eps-neighborhood radius as in DBSCAN algorithm there is no difference, because each point has four points within its Eps.

Since the most complex part of the algorithm is finding the k-nearest neighbors of each datapoint, a simple and efficient method is required for this. The original paper suggests a data structure called canopy to accomplish this task. In our project we decided to implement nearest neighbor queries by using a KD-tree [7], which is a data structure that also provides efficient nearest neighbor queries.

After we have found the k-nearest neighbors for each datapoint and, based on them, calculated the local density for each datapoint, it is possible to remove outliers based on a threshold. To detect the outliers we assign an outlying factor for each data point, this value is based on the local density. The outlying factor for a point *x* is given by the following equation:

$$OF(x) = \frac{DEN(x)}{MAX(DEN(x_1), DEN(x_2), \ldots, DEN(x_n))}$$

The numerator represents the local density at the point x which was computed by Equation (3), the denominator represents the local density of the lowest density point in the dataset. So the outlying factor for a point x is ranging from 0 to 1, as the outlying factor becomes closer to one, the higher the probability for the point to be an outlier. The outlier threshold is the only required input parameter for DCBOR and the algorithm supports the user in determining an appropriate value for it. The algorithm divides the interval [0,1] into 20 sub-intervals, and determines the count of points in each sub-interval. By examining this information the user can determine appropriate value for the outlier threshold. All points that have an outlying factor larger than the input value are discarded as outliers.

The clustering process is a middle ground between the single link algorithm and DBSCAN, since in single link two points are merged in each step, but here all points at distance from the current point that satisfy the threshold are assigned to the current cluster. We can deduce a suitable value for the threshold from the nearest neighbors of each leftover datapoint after removing the outliers. We search for the maximum distance between a point and its first nearest neighbor. This distance is the ideal distance for the threshold (level of dissimilarity between clusters) according to the main idea of the single link algorithm. The entire algorithm goes as follows:

1. For each point p return the k-nearest neighbors in ascending order according to their distance to p.
2. Calculate the local density for each point p.
3. Sort the points by their local density, highest density points first (lower numerical value).

4. Remove the outliers from the dataset according to the input parameter.
5. Determine the threshold = maximum distance to the first nearest neighbor.
6. Take the point with the highest density that is not already clustered to be the starting point for the current cluster.
7. Expand the current cluster according to the threshold (going from the nearest neighbors and their neighbors, adding point not already clustered) until no point can be added to it.
8. Start a new cluster and repeat steps 6 and 7 until all points are clustered.

Since we used a KD-tree data structure in our project, instead of the canopy method suggested in the paper, the time complexity of our implementation is also a little different. The first step takes *O(n*log n)* time in average, since the KD-tree data structure facilitates nearest neighbor queries with an average time complexity of *O(log n)*. Step 3 also takes *O(n*log n)* time, when using quick sort for example. Steps 2, 4, 5 and the clustering all take linear time. Therefore the time complexity of the algorithm is *O(n*log n)*.

SNN

Shared nearest neighbor (SNN) is density based clustering algorithm developed by Ertöz, Steinbach and Kumar [4] to find clusters with different shapes, sizes, densities and in high dimensional data. SNN works similarly to DBSCAN, but it does not use Euclidean distance to define similarity and to find densities of points. In high dimensional data Euclidean distances become more uniform, making it more difficult to cluster. Also this does not allow DBSCAN to find clusters with different densities. Instead SNN defines similarity between points by the number of nearest neighbors these points share. For example if point *p1* is close to point *p2* and they are both close to a set of points, S, then their similarity is equal to the number of points in set S. Density is defined as the number of points that are similar to a point. This allows SNN to avoid problems with high dimensional data and also to identify clusters of different densities.

SNN expects 3 parameters as input. Parameter *k* is the neighborhood list size. If *k* is too small then even relatively uniform clusters will be broken up, if it is too big then smaller clusters will not be found. Parameter *MinPts* is core point density threshold, points that have at least *MinPts* similar points will be considered core points. Parameter *Eps* is threshold for link strength, weaker links will be removed.

The steps of the SNN clustering algorithm are as follows:

1. Compute the similarity matrix.
2. Sparsify the similarity matrix by keeping only the *k* most similar neighbors.
3. Construct the shared nearest neighbor graph from the sparsified similarity matrix.
4. Find the SNN density of each point. For this the number of points that have SNN similarity of *Eps* or greater to each point has to be counted.
5. Find the core points. Core points are the ones that have density of *Eps* or greater.
6. Form clusters from the core points. Core points that are within *Eps* of eachother are placed in the same cluster.
7. Discard all noise points. All non-core points that are not within radius of *Eps* of some core point are considered noise.
8. Assign all non-noise, non-core points to clusters.

Run-time complexity of this algorithm is $O(n^2)$, because similarity matrix has to be constructed, but in low dimensional data *k*-d tree or R* tree can be used to perform finding nearest neighbors in *O(n * log n)*. In high dimensions canopies technique could be used in many cases, this means that points are first cheaply partitioned into smaller, possibly overlapping, groups and then clustering these smaller groups individually is faster. Rest of the algorithm is linear in the number of points.

Implementation

Right from the beginning the decision was made to implement our clustering visualization application in the Java programming language. The main benefit of this that Java is platform independent, so our application can be deployed on different platforms like Windows, Linux and Mac without requiring any platform specific code to be written or recompiling of the entire application. This made it also possible to deploy the application with all it's dependencies in a single JAR file, which doesn't require any installation on the user's system.

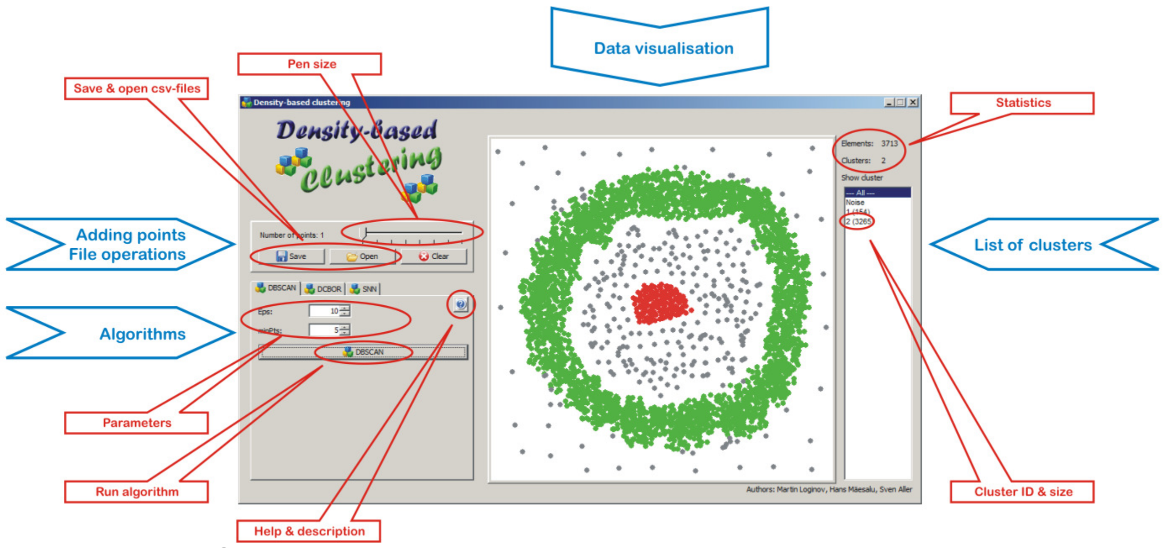
Application functionality

The main goal of the application is to give the user a "feel" of different density based clustering algorithms and how they work with different data distributions in identifying clusters. We chose to cluster 2-dimensional point data, because this type of data is the easiest to visualize, which in turn makes it easy to interpret the clustering results and make meaningful conclusions.

The application provides the following functionality to the user:

- **Enter the point data:** For this the user can just input the datapoints using the mouse by clicking on an area provided by the application. In order to make entering different types of datasets easier the user can choose to enter either a single datapoint at the time or to enter a randomly distributed mesh of point with a specified parameter. This last feature is intended to make entering large datasets less tedious.
- **Save/Load point data:** When the user has entered a dataset, he or she has the possibility of saving the layout of the datapoints in order to reuse them at a later time. This way it is convenient to reproduce clusterings on specific datasets without the need to reenter them every single time. Application allows to use data from any csv-file with integers, all points are fitted on the canvas.
- **Run 3 different density based clustering algorithms on input data**: The user has the choice of running either DBSCAN, DCBOR or SNN on the input datapoints. For each algorithm he or she has the possibility to fine tune the parameters of the specific algorithm and see the results of the clustering process visualized by representing different clusters with different colors. An additional feature for the user is the ability to visualize only a specific cluster or points classified as noise.
- **Get help info for different parameters:** The user also has the possibility of getting a short description displayed for each of the implemented algorithms and hints about the meaning of each of their parameters.

Graphical user interface


Density-based clustering

Number of points: 1

Eps: 10

minPts: 5

Elements: 3713

Clusters: 2

Show cluster

--- All ---
Noise
1 (154)
2 (3265)

Authors: Martin Loginov, Hans Mäesalu, Sven Aller

Figure 2. Graphical user interface.

Used libraries and technology

The only third-party library used in the implementation is the Java Machine Learning Library (Java-ML) [link]. Since this is an open source library, it is bundled together with the main application inside the JAR file. Java-ML was included, because it had an implementation of the KD-tree data structure, which the Java standard library is lacking. Using an already tried and tested implementation of this nontrivial data structure allowed us to focus on implementing the actual algorithms instead of spending the bulk of our time on code that was not actually the focus of our project.

The KD-tree structure was also chosen, because it could be used for both the DCBOR and the SNN algorithms. Implementing the canopy structure originally described in the paper for DCBOR was deemed too complex and time consuming and out of the scope of the project.

The GUI of the application was built using awt in WindowBuilder [link]. WindowBuilder is powerful bi-directional Java GUI WYSIWYG designer for creating Java applications: it allows graphically create new forms or visualize writed code. WindowBuilder is built as a plugin for Eclipse.

Algorithm comparison

To get an idea of how the results of DBSCAN, DCBOR and SNN compared to each other on different kinds of datasets, we created three different datasets shown on figure [ref]. These datasets were generated so we could see how well the different algorithms could identify clusters of different shape, density and how well they could find meaningful clusters out of noisy data.

Since there are many possible combinations for the different parameters of the algorithms, we only present here the parameters for each algorithm that to us seemed to give the most natural clustering results. Indeed the main goal of our project was to let the users experiment with the parameters themselves, not to optimize the algorithms or the results.

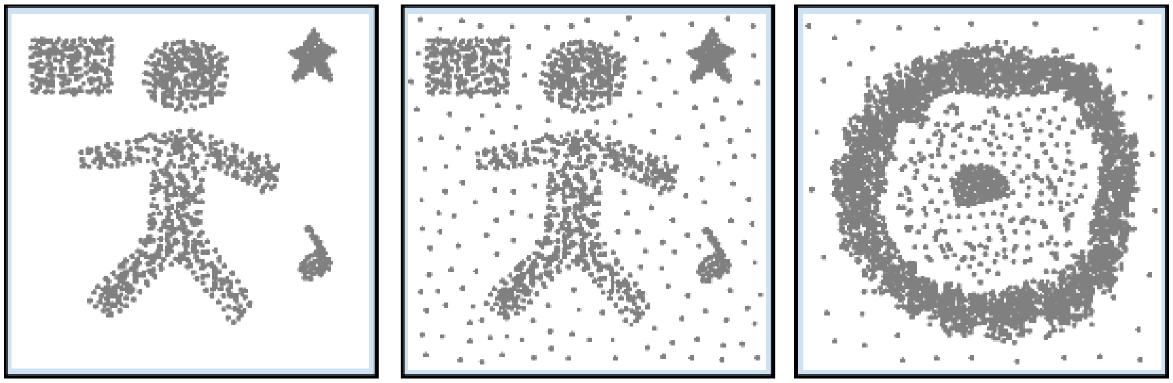


Figure 3. Datasets used for algorithms comparison.

Clusters of different shapes

All density based clustering algorithms we chose for our project should be able to identify clusters of different shapes and sizes. To test how well each algorithm would perform with different shapes we generated test dataset with several relatively complex shapes and ran each algorithm on it.

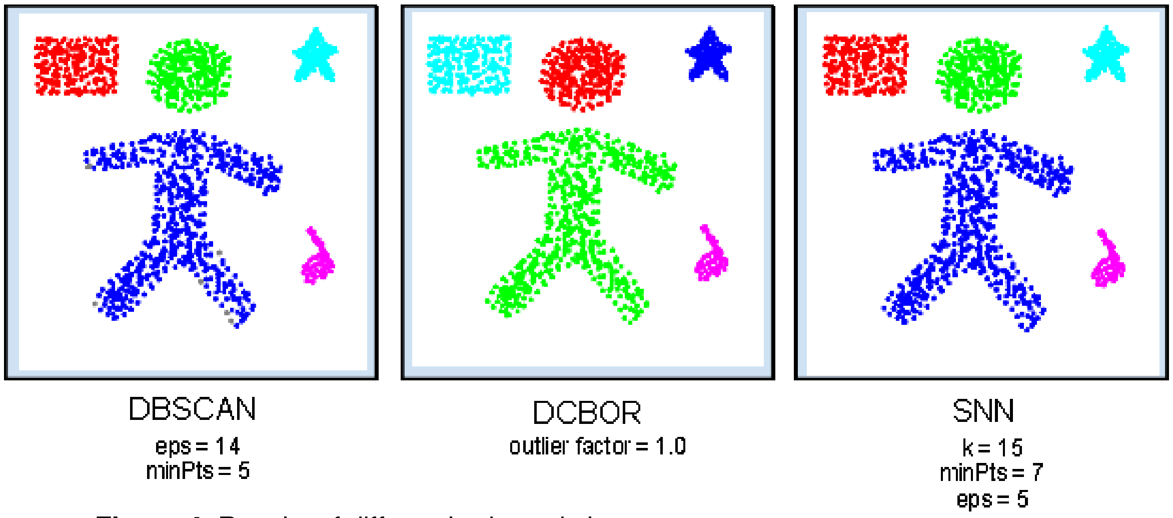


Figure 4. Results of differently shaped clusters test.

As can be seen on figure 4, all algorithms can identify all shapes perfectly without any errors.

Noise handling

All density based clustering algorithms we chose for our project should be able to identify noise points and not include it in any of the clusters. To test it we took the same dataset as in different shapes test, added some noise and then ran all algorithms on it.

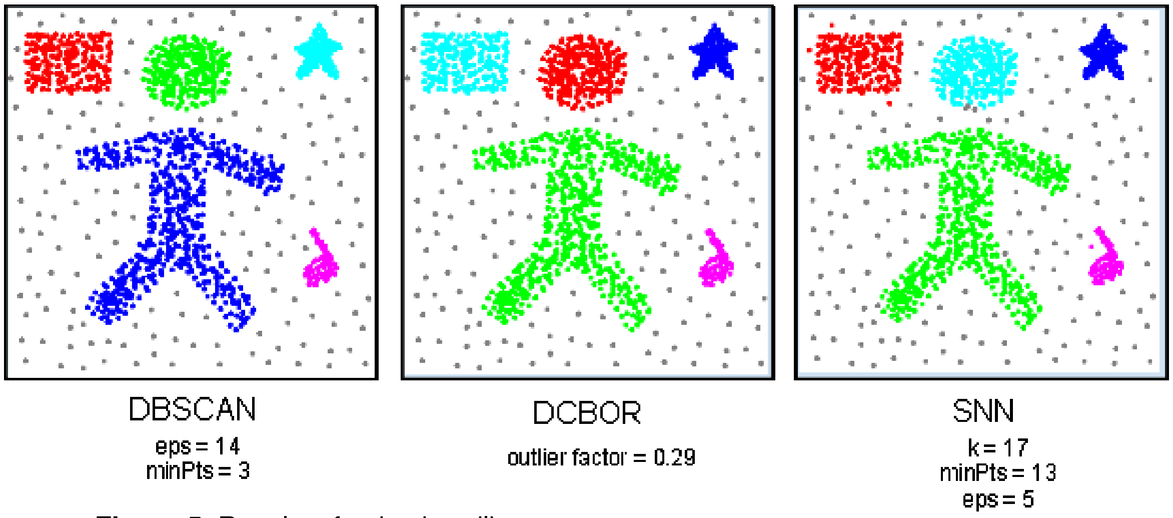


Figure 5. Results of noise handling test.

As can be seen from figure 5 all the algorithms perform very well with some parameter tuning, identifying the natural shapes even in the presence of noise. DBSCAN and DCBOR perfectly found all the clusters and did not include noise in the clusters. SNN found all the clusters, but it excluded some points, that we would have considered part of these clusters, and it included couple of noise points in clusters.

Clusters with different densities

One of the main problems with standard DBSCAN is its inability to identify clusters with different densities, DCBOR and SNN should not have this problem. To test how each algorithm would handle clusters with different densities we generated test dataset with 3 different clusters inside each other and ran all algorithms on it.

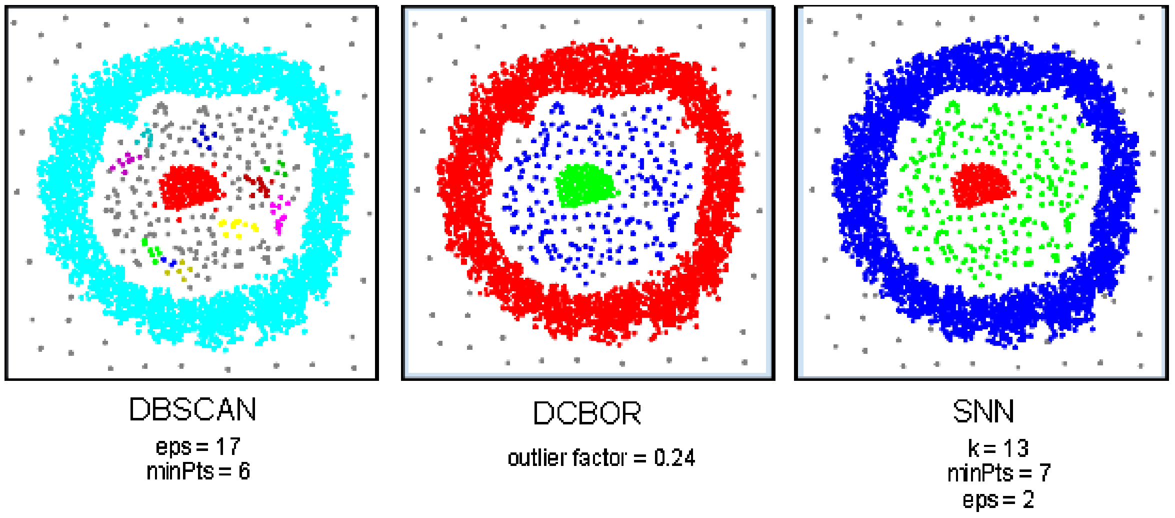


Figure 6. Results of different densities clusters test.

Here we begin to see some noteworthy differences between the algorithms. Here we hoped to see that the algorithms could distinguish between the circular clusters with different densities and discard the noise around the outermost circular cluster.

After some very careful parameter tuning the final results can be seen on figure x. DBSCAN has clearly the worst results here. It could identify the innermost circular cluster and the outermost circular cluster, which are both roughly of the same density, but found many different small clusters among the “middle” circular cluster.

DCBOR was somewhat more successful identifying the two dense clusters and even most of the middle cluster, but classified some points in the middle as noise.

SNN identified all the clusters with different densities perfectly.

Conclusions

The main goal of the project was achieved - the Java clustering application was implemented with all the initially planned functionality and it is working without any significant bugs. An overview of three different density-based clustering algorithms was also given.

During the comparison of the three algorithms a few things became clear: the less parameters an algorithms has, the easier it is to get any meaningful results out of it. For example in the case of DCBOR, while it did not always return the most "natural" results, finding the best possible results for it was a rather quick process. Thanks to the density distribution table of DCBOR it is also very easy to identify and discard noise points. SNN clustering algorithm, on the other hand, may return nearly perfect results with very complex datasets, but because there are three different parameters, that all affect the end result, and there is no easy way to determine these parameters, other than trial and error, then getting good results can take several attempts and much longer time.

Application source code is publicly available in Git repository at https://github.com/hansm/dm2011

References

1. Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowe. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. http://www2.cs.uh.edu/~ceick/7363/Papers/dbscan.pdf
2. DBSCAN - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/DBSCAN
3. Hencil Peter, J; Antonysamy, A. An Optimised Density Based Clustering Algorithm. International Journal of Computer Applications. Volume 6– No.9, September 2010. http://www.ijcaonline.org/volume6/number9/pxc3871445.pdf
4. Levent Ertöz, Michael Steinbach, Vipin Kumar. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. http://www-users.cs.umn.edu/~ertoz/snn/snn_siam03_final.pdf
5. Java Machine Learning Library (Java-ML). http://java-ml.sourceforge.net/
6. A. M. Fahim, G. Saake, A. M. Salem, F. A. Torkey and M. A. Ramadan. DCBOR: A Density Clustering Based on Outlier Removal. www.waset.org/journals/waset/v45/v45-31.pdf
7. k-d tree - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Kd-tree
8. Single-linkage clustering - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Single_link